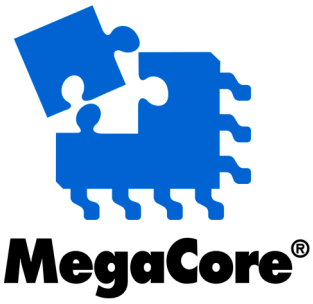


8B10B Encoder/Decoder

MegaCore Function User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

MegaCore Version: 7.2
Document Date: October 2007

Copyright © 2007 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

I.S. EN ISO 9001

UG-IPED8B10B-1.10

Contents

About This User Guide v
Revision History v
How to Contact Altera vi
Typographic Conventions 1–vii

Chapter 1. About This MegaCore Function
Release Information 1–1
Device Family Support 1–1
Features 1–2
General Description 1–2
OpenCore Plus Evaluation 1–2
Performance 1–3

Chapter 2. Getting Started
Design Flow 2–1
8B10B Encoder /Decoder Walkthrough 2–2
Create a New Quartus II Project 2–2
Launch MegaWizard Plug-in Manager 2–4
Parameterize 2–6
Set Up Simulation 2–7
Generate Files 2–8
Set Constraints 2–10
Simulate the Design 2–12
IP Functional Simulation Model 2–12
Compile the Design 2–13
Program a Device 2–13
Set Up Licensing 2–13

Chapter 3. Specifications
Functional Description 3–1
Disparity 3–1
Generic Framing Procedure 3–2
Character Codes 3–3
Encoder 3–4
Decoder 3–8
OpenCore Plus Time-Out Behavior 3–9
Parameters 3–10
Signals 3–10

About This User Guide

Revision History

The table below displays the revision history for the chapters in this user guide.

Chapter	Date	Version	Changes Made
All	October 2007	7.2	• Corrected error in text: the decoder (not the encoder) asserts `kerr` upon receiving an invalid code. • Updated discussion of Encoder in the Specifications chapter and timing diagrams Figure 3–5 and Figure 3–6.
All	May 2007	7.1	• Added support for Arria™ GX device family • Maintenance release; updated product release information
1	December 2006	7.0	• Added support for Cyclone III devices.
	April 2006	1.6.1	• Updated release information and device family support tables.
	October 2005	1.6.0	• Updated release information and device family support tables.
	June 2004	1.5.0	• Updated release information and device family support tables. • Updated performance information.
	February 2004	1.4.0	• Updated release information and device family support tables. • Added OpenCore Plus description. • Updated performance information.
2	December 2006	6.1	• Updated screen shots to match new version.
	April 2006	1.6.1	• Updated walkthrough instructions.
	October 2005	1.6.0	• Updated system requirements
	June 2004	1.5.0	• Update system requirements • Updated the instructions to obtain the 8B10B Encoder/Decoder MegaCore® function.
	February 2004	1.4.0	• Added Linux instructions. • Added IP Toolbench instructions. • Added IP functional simulation models information.

Chapter	Date	Version	Changes Made
3	April 2006	1.6.1	• Added text to Encoded Latency description. • Added "Encoder Timing Diagram—One Cycle Latency" on page 3–7. • Made corrections to `clk` signal description. • Removed `Device family` parameter and added `Registered inputs/outputs` parameter.
	October 2005	1.6.0	• Removed Mercury™ devices from the `Device family` parameter options, and added the HardCopy® II and Stratix® II GX families.
	June 2004	1.5.0	• No changes.
	February 2004	1.4.0	• Added running disparity error output (`rderr`) description. • Added 10B_ERR special code description. • Added OpenCore Plus time-out behavior description. • Added Parameters description (table). • Updated Signals tables.

How to Contact Altera

For the most up-to-date information about Altera® products, go to the Altera world-wide website at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

Information Type	Contact *(1)*
Technical support	www.altera.com/mysupport/
Technical training	www.altera.com/training/
Technical training services	custrain@altera.com
Product literature	www.altera.com/literature
Product literature services	literature@altera.com
FTP site	ftp.altera.com

Note:
(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: "Typographic Conventions."
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input.` Active-low signals are denoted by suffix `n`, for example, `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (for example, the VHDL keyword `BEGIN`), as well as logic function names (for example, `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ● •	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or the user's work.
WARNING	A warning calls attention to a condition or possible situation that can cause injury to the user.
↵	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.

1. About This MegaCore Function

Release Information

Table 1–1 provides information about this release of the Altera® 8B10B Encoder/Decoder MegaCore® function.

Table 1–1. 8B10B Encoder/Decoder MegaCore Function Release Information

Item	Description
Version	7.2
Release Date	October 2007
Ordering Code	IP-ED8B10B
Product ID	0079
Vendor ID	6AF7

Device Family Support

MegaCore functions provide either full or preliminary support for target Altera device families:

- *Full support* means the MegaCore function meets all functional and timing requirements for the device family and may be used in production designs.
- *Preliminary support* means the MegaCore function meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution.

Table 1–2 shows the level of support offered by the 8B10B Encoder/Decoder MegaCore function to each Altera device family.

Table 1–2. Device Family Support (Part 1 of 2)

Device Family	Support
Arria™ GX	Preliminary
Cyclone®	Full
Cyclone II	Full
Cyclone III	Preliminary
HardCopy® II	Full

Table 1–2. Device Family Support (Part 2 of 2)	
Device Family	**Support**
HardCopy Stratix®	Full
Stratix	Full
Stratix II	Full
Stratix II GX	Full
Stratix III	Preliminary
Stratix GX	Full
Other device families	No support

Features

- Support for Arria™ GX device family
- 8b/10b encoding and decoding
- Cascaded encoding and decoding
- Industry compatible special character coding
- Easy-to-use IP MegaWizard® interface
- Support for OpenCore Plus evaluation
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators

General Description

Encoders and decoders are used for physical layer coding for Gigabit Ethernet, Fibre Channel, and other applications. The 8b/10b encoder takes byte inputs, and generates a direct current (DC) balanced stream (equal number of 1s and 0s) with a maximum run length of 5. Some of the individual 10-bit codes will have an equal number of 1s and 0s, while others will have either four 1s and six 0s, or, six 1s and four 0s. In the latter case, the disparity between 1s and 0s is used as an input to the next 10-bit code generation, so that the disparity can be reversed, and maintain an overall balanced stream. For this reason, some 8-bit inputs have two valid 10-bit codes, depending on the input disparity.

The Altera 8B10B Encoder/Decoder is a compact, high performance MegaCore function capable of encoding and decoding in multi-gigabit applications.

OpenCore Plus Evaluation

With Altera's free OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction within your system

- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily
- Generate time-limited device programming files for designs that include megafunctions
- Program a device and verify your design in hardware

You only need to purchase a license for the megafunction when you are completely satisfied with its functionality and performance, and want to take your design to production.

For more information on OpenCore Plus hardware evaluation using the 8B10B Encoder/Decoder see "OpenCore Plus Time-Out Behavior" on page 3–9 and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Performance

Table 1–3, Table 1–4, and Table 1–5 show the resource utilization and performance of some 8B10B Encoder/Decoder MegaCore functions. These results were obtained using the Quartus® II software version7.1 for the following devices:

- Cyclone II (EP2C35F484C6)
- Cyclone III (EP3C80F780C6)
- Stratix II (EP2S30F484C3)
- Stratix III (EP3SE110F780C2)

Table 1–3. Resource Utilization and Performance (Cyclone II, Cyclone III)

Device	Parameters		LEs	f_{MAX} (MHz) *(1) (2)*
	Mode of Operation	Register Inputs/Outputs		
Cyclone II	Encoder	On	100	250
	Encoder	Off	107	454
	Decoder	–	131	403
Cyclone III	Encoder	On	100	250
	Encoder	Off	107	454
	Decoder	–	131	403

Notes to Table 1–3:

(1) f_{MAX} is for non-cascaded encoders/decoders.

(2) These results were obtained with the auto-ROM replacement feature disabled in the Quartus II software. Enabling this feature produces a smaller but slower MegaCore function.

Table 1–4. Resource Utilization and Performance (Stratix II)

Device	Parameters		Combinational ALUTs	Logic Registers	f_{MAX} (MHz) *(1) (2)*
	Mode of Operation	Register Inputs/Outputs			
Stratix II	Encoder	On	61	51	444
	Encoder	Off	68	13	585
	Decoder	–	55	33	447

Notes to Table 1–4:

(1) f_{MAX} is for non-cascaded encoders/decoders.

(2) These results were obtained with the auto-ROM replacement feature disabled in the Quartus II software. Enabling this feature produces a smaller but slower MegaCore function.

Table 1–5. Resource Utilization and Performance (Stratix III)

Device	Parameters		Combinational ALUTs	Logic Registers	f_{MAX} (MHz) *(1) (2)*
	Mode of Operation	Register Inputs/Outputs			
Stratix III	Encoder	On	60	51	510
	Encoder	Off	68	13	675
	Decoder	–	55	33	520

Notes to Table 1–5:

(1) f_{MAX} is for non-cascaded encoders/decoders.

(2) These results were obtained with the auto-ROM replacement feature disabled in the Quartus II software. Enabling this feature produces a smaller but slower MegaCore function.

2. Getting Started

Design Flow

To evaluate the 8B10B Encoder/Decoder MegaCore® function using the OpenCore Plus feature include these steps in your design flow:

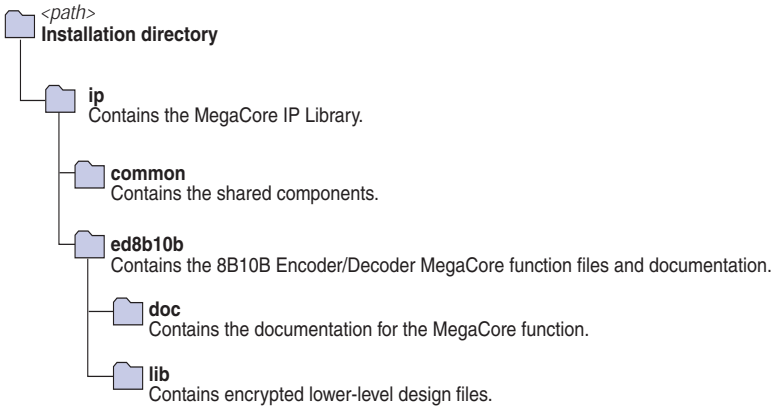
1. Obtain and install the 8B10B Encoder/Decoder MegaCore Function.

The 8B10B Encoder/Decoder MegaCore Function is part of the MegaCore IP Library, which is distributed with the Quartus® II software and downloadable from the Altera website, **www.altera.com**.

For system requirements and installation instructions, refer to *Quartus II Installation & Licensing for Windows* or *Quartus II Installation & Licensing for UNIX & Linux Workstations* on the Altera website.

Figure 2–1 shows the directory structure after you install the 8B10B Encoder/Decoder, where *<path>* is the installation directory. The default installation directory on Windows is **c:\altera\71**; on UNIX and Linux it is **/opt/altera/71**.

Figure 2–1. Directory Structure



2. Create a custom variation of the 8B10B Encoder/Decoder MegaCore Function.

3. Implement the rest of your design using the design entry method of your choice.

4. Use the IP functional simulation model to verify the operation of your design.

For more information on IP functional simulation models, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in Volume 3 of the *Quartus II Handbook*.

5. Use the Quartus II software to compile your design.

You can also generate an OpenCore Plus time-limited programming file, which you can use to verify the operation of your design in hardware.

6. Purchase a license for the 8B10B Encoder/Decoder MegaCore Function.

After you have purchased a license for the 8B10B Encoder/Decoder MegaCore Function, follow these additional steps:

1. Set up licensing.
2. Generate a programming file for the Altera® device(s) on your board.
3. Program the Altera device(s) with the completed design.

8B10B Encoder /Decoder Walkthrough

This walkthrough shows you how to create an 8B10B Encoder/Decoder MegaCore function using the MegaWizard interface and the Quartus II software. After generating a custom variation of the 8B10B Encoder/Decoder MegaCore function, you can incorporate it into your overall project.

This walkthrough consists of these steps:

- Create a New Quartus II Project
- Launch MegaWizard Plug-in Manager
- Parameterize
- Set Up Simulation
- Generate Files
- Set Constraints

Create a New Quartus II Project

You need to create a new Quartus II project with the **New Project Wizard**, which specifies the working directory for the project, assigns the project name, and designates the name of the top-level design entity.

To create a new project follow these steps:

1. Choose **Programs > Altera > Quartus II** <*version*> (Windows Start menu) to run the Quartus II software. Alternatively, you can use the Quartus II Web Edition software.

2. Choose **New Project Wizard** (File menu).

3. Click **Next** in the **New Project Wizard Introduction** page (the introduction does not display if you turned it off previously).

4. In the **New Project Wizard: Directory, Name, Top-Level Entity** page, enter the following information:

   a. Specify the working directory for your project. For example, this walkthrough uses the directory:

      c:\altera\projects\ed8b10b_project

   b. Specify the name of the project. This walkthrough uses the project name:

      ed8b10b_example

   The Quartus II software automatically specifies a top-level design entity that has the same name as the project. Do not change it.

5. Click **Next** to close this page and display the **New Project Wizard: Add Files** page.

   When you specify a directory that does not already exist, a message asks if the specified directory should be created. Click **Yes** to create the directory.

6. If you installed the MegaCore IP Library in a different directory from where you installed the Quartus II software, you must add the user libraries:

   a. Click **User Libraries**.

   b. Type <*path*>\ip into the **Library name** box, where <*path*> is the directory in which you installed the 8B10B Encoder/Decoder MegaCore Function.

   c. Click **Add** to add the path to the Quartus II project.

d. Click **OK** to save the library path in the project.

7. Click **Next** to close this page and display the **New Project Wizard: Family & Device Settings** page.

8. On the **New Project Wizard: Family & Device Settings** page, choose the target device family in the **Family** list.

9. The remaining pages in the **New Project Wizard** are optional. Click **Finish** to complete the Quartus II project.

You have finished creating your new Quartus II project.

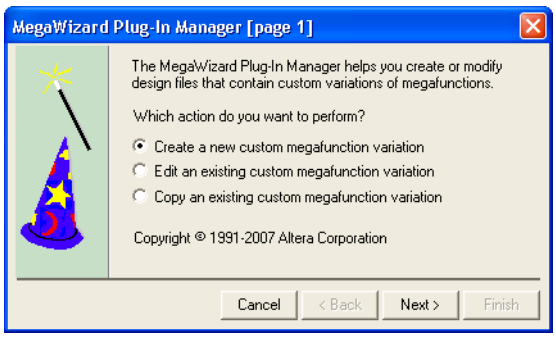
Launch MegaWizard Plug-in Manager

To launch the MegaWizard Plug-in Manager in the Quartus II software, follow these steps:

1. Start the MegaWizard® Plug-In Manager by choosing the **MegaWizard Plug-In Manager** command (Tools menu). The **MegaWizard Plug-In Manager** dialog box displays (see Figure 2–2).

Refer to the Quartus II Help for more information on how to use the MegaWizard Plug-In Manager.

Figure 2–2. MegaWizard Plug-in Manager


MegaWizard Plug-In Manager [page 1]
The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.
Which action do you want to perform?
Create a new custom megafunction variation
Edit an existing custom megafunction variation
Copy an existing custom megafunction variation
Copyright © 1991-2007 Altera Corporation
Cancel
< Back
Next >
Finish

2. Specify that you want to create a new custom megafunction variation and click **Next**.

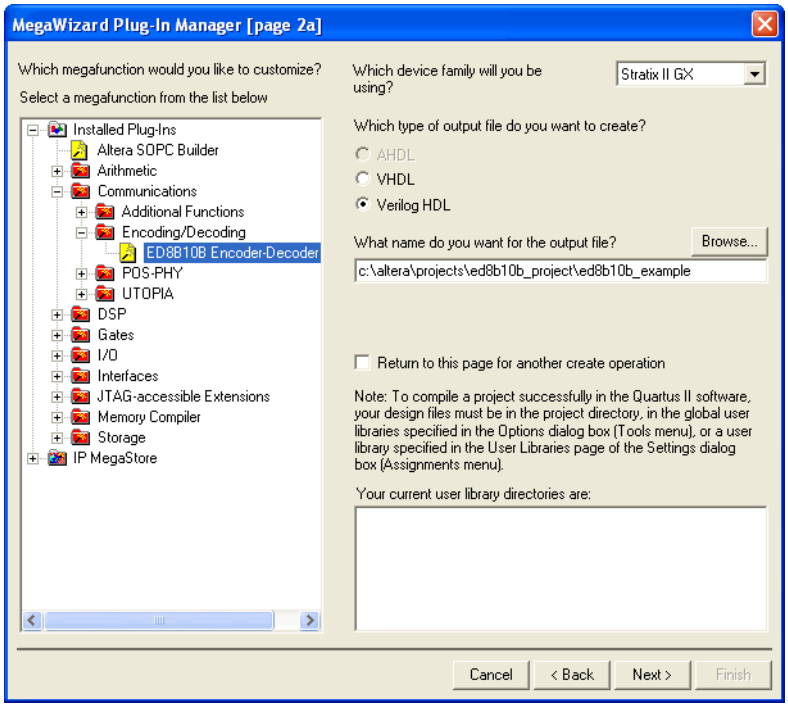
3. Expand the **Communications** > **Encoding/Decoding** directory, then click **8B10B Encoder-Decoder v7.1**.

4. Choose the device family you want to use for this MegaCore function, for example **Stratix II GX**.

5. Select the output file type for your design; the MegaWizard interface supports VHDL and Verilog HDL.

6. The MegaWizard Plug-In Manager shows the project path that you specified in the **New Project Wizard**. Append a variation name for the MegaCore function output files *<project path>**<variation name>*. Figure 2–3 shows the MegaWizard Plug-In Manager after you have made these settings.

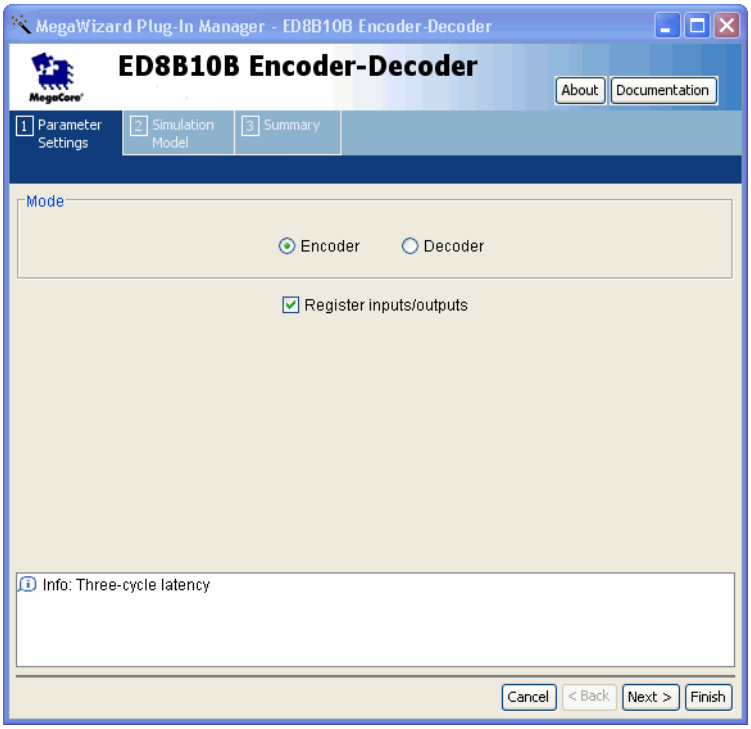
Figure 2–3. Select the MegaCore Function


MegaWizard Plug-In Manager [page 2a]
Which megafunction would you like to customize?
Select a megafunction from the list below
Installed Plug-Ins
Altera SOPC Builder
Arithmetic
Communications
Additional Functions
Encoding/Decoding
ED8B10B Encoder-Decoder
POS-PHY
UTOPIA
DSP
Gates
I/O
Interfaces
JTAG-accessible Extensions
Memory Compiler
Storage
IP MegaStore
Which device family will you be using?
Stratix II GX
Which type of output file do you want to create?
AHDL
VHDL
Verilog HDL
What name do you want for the output file?
Browse...
c:\altera\projects\ed8b10b_project\ed8b10b_example
Return to this page for another create operation
Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the Options dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu).
Your current user library directories are:
Cancel
< Back
Next >
Finish

7. Click **Next** to display the **Parameter Settings** page for the 8B10B Encoder/Decoder MegaCore Function (see Figure 2–4).

You can change the page that the MegaWizard Plug-In Manager displays by clicking **Next** or **Back** at the bottom of the dialog box. You can move directly to a named page by clicking the **Parameter Settings**, **Simulation Model**, or **Summary** tab.

Figure 2–4. Parameters



Parameterize

To parameterize your MegaCore function, follow these steps:

1. Select the mode of operation, either **Encoder** or **Decoder**
2. If you selected **Encoder**, turn on the **Register inputs/outputs** check box for a three-cycle latency, or turn off the **Register inputs/outputs** check box for a single-cycle latency.

The **Decoder** always has registered inputs and outputs.

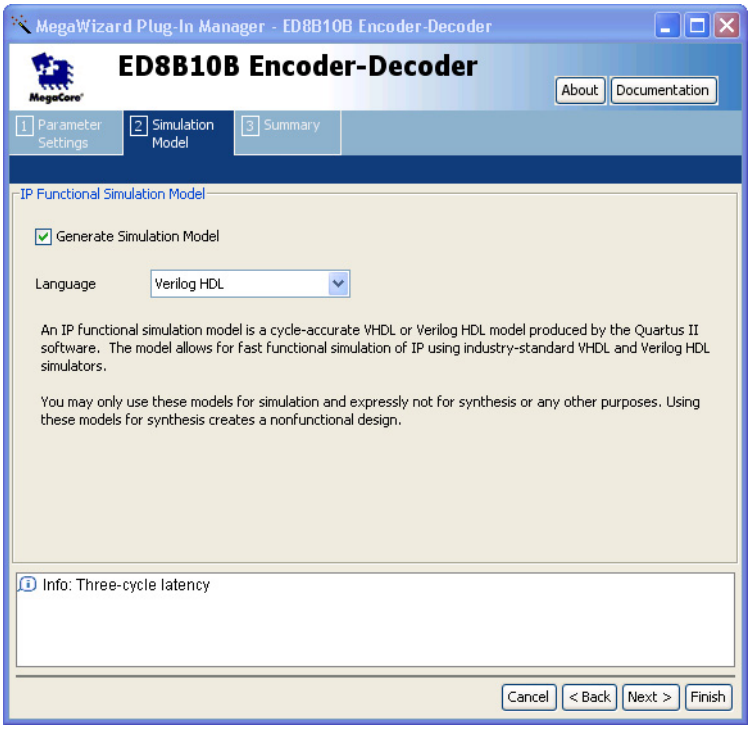
3. Select the mode of operation, either **Encoder** or **Decoder**.

4. If you selected **Encoder**, turn on the **Register inputs/outputs** check box for a three cycle latency, or turn off the **Register inputs/outputs** check box for a single cycle latency.

The **Decoder** always has registered inputs and outputs.

5. Click **Next** (or the **Simulation Model** tab) to display the simulation setup page (see Figure 2–5).

Figure 2–5. Simulation Model


MegaWizard Plug-In Manager - ED8B10B Encoder-Decoder
ED8B10B Encoder-Decoder
About
Documentation
1 Parameter Settings
2 Simulation Model
3 Summary
IP Functional Simulation Model
Generate Simulation Model
Language
Verilog HDL
An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software. The model allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.
You may only use these models for simulation and expressly not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.
Info: Three-cycle latency
Cancel
< Back
Next >
Finish

Set Up Simulation

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model file produced by the Quartus II software. The model allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

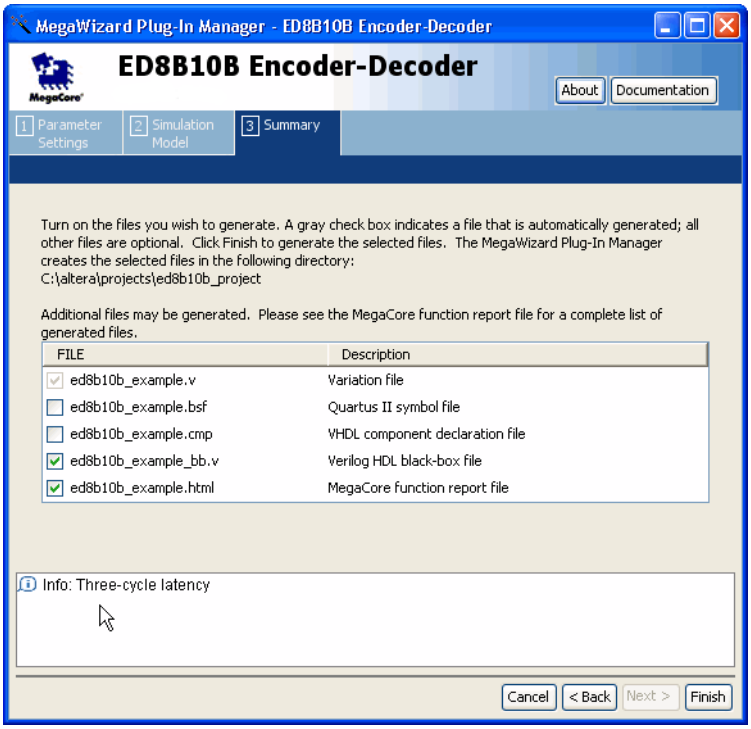
CAUTION

You may only use these models for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

To generate an IP functional simulation model for your MegaCore function, follow these steps:

1. Turn on **Generate Simulation Model**.
2. Choose the language you want from the **Language** list.
3. Click **Next** (or the **Summary** page) to display the summary page (see Figure 2–6).

Figure 2–6. Summary Page


MegaWizard Plug-In Manager - ED8B10B Encoder-Decoder
ED8B10B Encoder-Decoder
About
Documentation
1 Parameter Settings
2 Simulation Model
3 Summary
Turn on the files you wish to generate. A gray check box indicates a file that is automatically generated; all other files are optional. Click Finish to generate the selected files. The MegaWizard Plug-In Manager creates the selected files in the following directory:
C:\altera\projects\ed8b10b_project
Additional files may be generated. Please see the MegaCore function report file for a complete list of generated files.
FILE
Description
ed8b10b_example.v
Variation file
ed8b10b_example.bsf
Quartus II symbol file
ed8b10b_example.cmp
VHDL component declaration file
ed8b10b_example_bb.v
Verilog HDL black-box file
ed8b10b_example.html
MegaCore function report file
Info: Three-cycle latency
Cancel
< Back
Next >
Finish

Generate Files

You can use the check boxes in the Summary page to enable or disable the generation of specified files. A gray checkmark indicates a file that is automatically generated; a red checkmark indicates an optional file.

You can click **Back** to display the previous page or click **Parameters Setting**, **Simulation Library**, or **Summary Page**, if you want to change any of the MegaWizard options.

To generate the files, follow these steps:

1. Turn on the files you want to generate (see Figure 2–6).
2. To generate the specified files and close the MegaWizard Plug-in Manager, click **Finish**.

   The generation phase may take several minutes to complete.

   When the file generation is complete, you can go to the project directory and view a list of generated files in the file <variation name>.**html** (the files that are listed in the following table).

   Table 2–1 describes the generated files and other files that may be in your project directory. The names and types of files specified in the summary vary based on whether you created your design with VHDL or Verilog HDL

Table 2–1. Generated Files Note (1) (Part 1 of 2)

Filename (2)	Description
<*variation name*>**.bsf**	Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
<*variation name*> . **cmp**	VHDL component declaration file
<*variation name*>**.html**	The MegaCore function report file.
<*variation name*>**.v**	A MegaCore function variation file, which defines a Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
<*variation name*>**.vo**	Verilog HDL IP functional simulation model.
<*variation namez*>**_bb.v**	Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.
<*variation name*>**_constraints.tcl**	Tool command language (tcl) script used to set constraints.
<*variation name*>**_enc8b10b.ocp**	An OpenCore Plus file, needed for time-limited or tethered hardware evaluation.
<*variation name*>**_enc8b10b.v**	Verilog HDL RTL for this MegaCore function variation.
<*variation name*>**_run_modelsim.tcl**	A Tcl script to automate the process of running the provided demo testbench with the IP functional simulation model.

Table 2–1. Generated Files Note (1) (Part 2 of 2)	
Filename *(2)*	**Description**
<*variation name*>**_tb.v**	A Verilog HDL module with the top-level demo testbench for the core.

Notes to Table 2–1:
(1) These files are variation dependent, some may be absent or their names may change.
(2) <*variation name*> is a prefix variation name supplied automatically by the MegaWizard interface.
(3) If you choose the decoder mode, the file name is <*variation name*>**_dec8b10b**.

Set Constraints

The 8B10B Encoder/Decoder MegaCore function variations include a tool command language (Tcl) script. Use this Tcl script to constrain your design.

To run the Tcl script in the Quartus II software, in a Win32 operating system, follow either of these sets of steps:

1. Select **TCL Scripts** (Tools menu)
2. Select the applicable Tcl file for your variation:

   <*variation name*>**_constraints. tcl**

3. Click **Run**.

   or

1. Click on **Tcl Console** under **Utility_Windows** (View menu)
2. In the Tcl console window, type:

```
source <variation name>_constraints.tcl
```

To run the Tcl script in a UNIX or Linux operating system terminal, type:

```
cd..<project_directory>

quartus_sh -t <variation name>_constraints.tcl
```

Depending on the type of constraints applied by the Tcl script, analysis and synthesis may be run twice. For example, if hierarchy independent constraints are needed, the Tcl script runs analysis and synthesis before applying the constraints. Therefore, when you run a full compilation, after running the Tcl script, the analysis and synthesis are run a second time.

You can now integrate your custom MegaCore function variation into your design, simulate, and compile.

Simulate the Design

You can simulate your design using the generated VHDL and Verilog HDL IP functional simulation models.

For more information on IP functional simulation models, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in Volume 3 of the *Quartus II Handbook*.

Altera also provides a Verilog HDL demonstration testbench, including scripts to compile and run the demonstration testbench using a variety of simulators and models. This testbench demonstrates the typical behavior of an 8B10B MegaCore function, and how to instantiate a model in a design. The demonstration testbench does not perform any error checking.

For a complete list of models or libraries required to simulate the 8B10B Encoder/Decoder MegaCore function, refer to the **_run_modelsim.tcl** scripts provided with the demonstration testbench.

IP Functional Simulation Model

To use the demonstration testbench with IP functional simulation models in the ModelSim® simulator, follow these steps:

1. Start the ModelSim simulator.
2. From the ModelSim **File** menu, use **Change Directory** to change the working directory to the directory where you created your 8B10B Encoder/Decoder variation.
3. In the ModelSim **Transcript** window, execute the command `do<variation_name>_run_modelsim.tcl` which sets up the required libraries, compiles the netlist files, and runs the testbench. The ModelSim **Transcript** window displays messages from the testbench reflecting the results of the simulation.

In all cases, the testbench is in Verilog HDL, therefore a license to run mixed language simulations is required to run the testbench with the VHDL model.

Altera recommends that you disable the auto-ROM replacement feature in the Quartus II software. Enabling this feature produces a smaller but slower MegaCore function.

Compile the Design

You can use the Quartus II software to compile your design. Refer to Quartus II Help for instructions on compiling your design.

Program a Device

After you have compiled your design, program your targeted Altera device, and verify your design in hardware.

With Altera's free OpenCore Plus evaluation feature, you can evaluate the 8B10B Encoder/Decoder MegaCore function before you purchase a license. OpenCore Plus evaluation allows you to generate an IP functional simulation model and produce a time-limited programming file.

For more information on IP functional simulation models, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in Volume 3 of the *Quartus II Handbook Version 7.1*.

For more information on OpenCore Plus hardware evaluation using the 8B10B Encoder/Decoder, see "OpenCore Plus Time-Out Behavior" on page 3–9 and *AN 320: OpenCore Plus Evaluation of Megafunctions.*

Set Up Licensing

You need to purchase a license for the MegaCore function only when you are completely satisfied with its functionality and performance, and want to take your design to production.

After you purchase a license for 8B10B Encoder/Decoder, you can request a license file from the Altera website at **www.altera.com/licensing** and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative.

3. Specifications

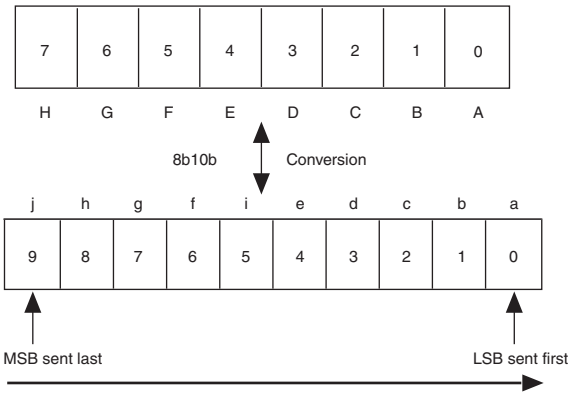
Functional Description

The MegaCore® function consists of an encoder (ENC8B10B) and a decoder (DEC8B10B). See Figure 3–2 on page 3–3. The encoder encodes one 8-bit byte of data into a 10-bit transmission code, and the decoder decodes a 10-bit code into one 8-bit byte of data. Figure 3–1 illustrates the bidirectional conversion process.

The eight input bits are named A, B, C, D, E, F, G, H. Bit A is the least significant bit (LSB), and bit H is the most significant bit (MSB). They are split into two groups: The five-bit group A, B, C, D, E, and the three-bit group F, G, H.

The coded bits are named a, b, c, d, e, i, f, g, h, j (the order is not alphabetical). These bits are also split into two groups: the six-bit group a, b, c, d, e, i, and the four-bit group f, g, h, j.

Figure 3–1. 8b10b Conversion



In bit serial transmission, the LSB is usually transmitted first, while the MSB is usually transmitted last.

Disparity

Disparity is the difference between the number of 1s and 0s in the encoded word.

- Neutral disparity indicates the number of 1s and 0s are equal.
- Positive disparity indicates more 1s than 0s.
- Negative disparity indicates more 0s than 1s.

The MegaCore function is designed to maintain a neutral average disparity. Average disparity determines the direct current (DC) component of a serial line. Running disparity is a record of the cumulative disparity of every encoded word, and is tracked by the encoder. To guarantee neutral average disparity, a positive running disparity must be followed by neutral or negative disparity; a negative running disparity must be followed by neutral or positive disparity.

The running disparity error output (`rderr`) is asserted when any of the following rules apply:

- The current running disparity is positive and the 6-bit group has more ones than zeros or is 111000
- The current running disparity is negative and the 6-bit group has more zeros than ones or is 000111
- The running disparity after 6-bit group is positive and the 4-bit group has more ones than zeros or is 1100
- The running disparity after 6-bit group is negative and the 4-bit group has more zeros than ones or is 0011

`rderr` is asserted for some invalid 10-bit codes and not for others, strictly based on the rules stated above. The computation of `rderr` is completely independent of that of the special control character error (`kerr`) signal.

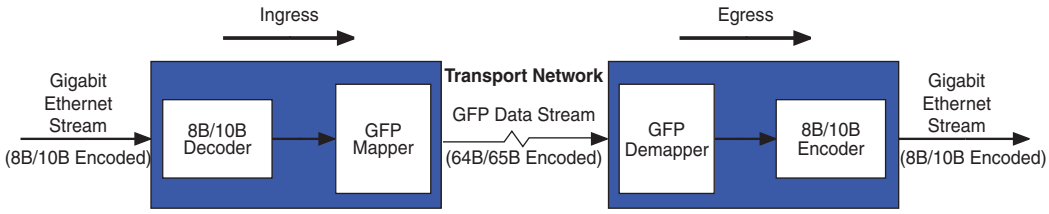
A 10-bit code that corresponds to a valid encoding but that has the wrong disparity—though technically an invalid code—does not cause the `kerr` signal to be asserted. Only `rderr` is asserted.

For details on running disparity rules, refer to the *IEEE 802.3z specification, paragraph 36.2.4.4*.

Generic Framing Procedure

The 8B10B Encoder/Decoder MegaCore function can be used within generic framing procedure (GFP) applications. See Figure 3–2 on page 3–3 for an example.

Figure 3–2. 8B10B Encoder/Decoder GFP Typical Application



On ingress to the transport network, if the decoder receives an unrecognized codeword, such as an illegal codeword or a legal codeword with a running disparity error, it asserts the `kerr` or `rderr` signals respectively. By asserting these error signals, the decoder indicates to the mapper that an invalid codeword has been received, the mapper then generates a special control character, the 10B_ERR code. In addition, the mapper remaps the 8B/10B codewords into 64B/65B codewords before sending the data to the transport network.

On egress from the transport network, the demapper decodes the 64B/65B codewords and sends them to the 8B/10B encoder. When the encoder receives the 10B_ERR code, it sends out one of the two 10-bit illegal codewords with neutral disparity: 001111 0001(RD-) or 110000 1110(RD+), depending on the running disparity.

Character Codes

In addition to 256 data characters, the 8b/10b code defines thirteen out-of-band indicators, also called special control characters. The 256 data characters are named D*x.y*, and the special control characters are named K*x.y*—except for the special code 10B_ERR (see Table 3–1 on page 3–4). The *x* value corresponds to the five-bit group, and the *y* value to the three-bit group.

The special control characters indicate, for example, whether the data is idle, test data, or data delimiters. In applications where encoded characters are transmitted bit-serially, the comma character (K28.5) is usually used for alignment purposes as its 10-bit code is guaranteed not to occur elsewhere in the encoded bit stream, except after K28.7 which is normally only sent during diagnostic.

Table 3–1 lists the special *K* codes used by the MegaCore function.

Table 3–1. Character Codes

10-Bit Special *K* Codes	Equivalent 8-Bit Codes
K28.0	8'b000_11100
K28.1	8'b001_11100
K28.2	8'b010_11100
K28.3	8'b011_11100
K28.4	8'b100_11100
K28.5 *(1)*	8'b101_11100
K28.6	8'b110_11100
K28.7	8'b111_11100
K23.7	8'b111_10111
K27.7	8'b111_11011
K29.7	8'b111_11101
K30.7	8'b111_11110
10B_ERR	8'b111_11111

Note to Table 3–1:
(1) K28.5 is a comma character used for alignment purposes, and to represent the IDLE code.

Encoder

To encode an 8-bit word, the 8-bit value must be applied to the `datain` inputs and the `ena` input must be asserted (active high).

When one of the thirteen special 10-bit codes is to be inserted, the equivalent 8-bit code is placed on the `datain` lines and the `kin` input is asserted. The MegaCore function performs error checking to ensure the out-of-band 8-bit code is valid. If not, the `kerr` output is asserted. See Table 3–1 for a list of the valid *K* codes.

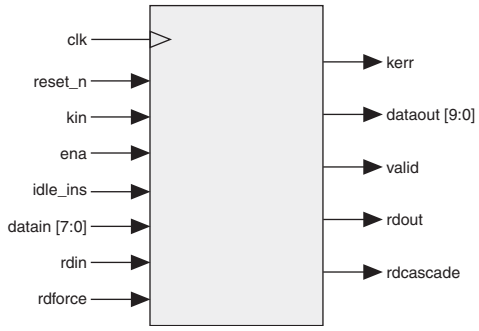
Although the 10B_ERR code is considered to be an invalid special character, it does not cause the `kerr` signal to be asserted.

Idle (K28.5) characters can be automatically inserted when `ena` is not asserted by asserting the `idle_ins` input.

The encoder encodes invalid characters in the same way it encodes Idle (K28.5) codes. The decoder treats invalid characters as Idle codes.

Figure 3–3 shows a block diagram of the encoder.

Figure 3–3. Encoder



Disparity

The running disparity can be forced to positive or negative, allowing the user to insert a special resynchronization pattern, or disparity errors.

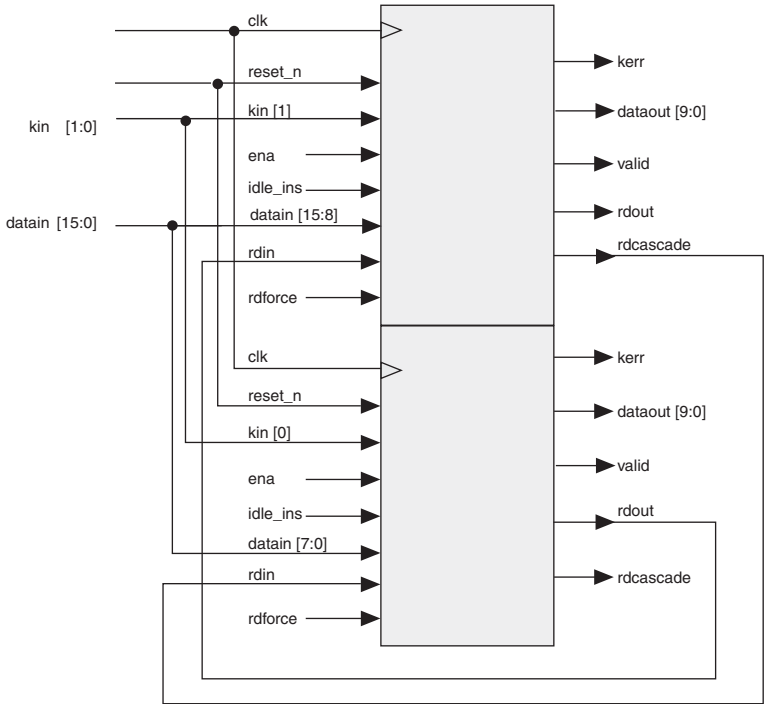
When the `rdforce` input is asserted, the value on the `rdin` port is assumed to be the current running disparity. Setting `rdin` to 0 forces the encoder to produce an encoded word with positive or neutral disparity. Setting `rdin` to 1 forces the encoder to produce an encoded word with negative or neutral disparity.

Cascaded Encoding

Two encoders can be cascaded to allow for 16-bit word encoding. The encoders are cascaded by connecting the `rdcascade` output of the most significant byte (MSByte) encoder to the `rdin` input of the least significant byte (LSByte) encoder, and by connecting the `rdout` output of the LSByte encoder to the `rdin` input of the MSByte encoder. These connections ensure proper running disparity computation. The `rdforce` inputs must be asserted (active high) for the encoders to take into account the value on the `rdin` inputs, rather than use their internally generated running disparity. Both `ena` inputs must be high or low at the same time. The `kin [1]` signal relates to `datain[15:8]`, and `kin[0]` relates to `datain[7:0]`. Figure 3–4 on page 3–6 shows two encoders connected together to perform cascaded encoding.

If the encoded words are to be transmitted serially, the result of encoding `datain[15:8]` should be transmitted first.

Figure 3–4. Cascaded Encoding *Note (1)*



Note to Figure 3–4:
(1) The `ena`, `idle_ins`, and `rdforce` signals are set high (logic 1).

Encoding Latency

When the `register inputs/outputs` parameter is turned on, the encoder is pipelined, thus it takes three clock cycles for a character to be encoded. The encoded value—corresponding to the values of `datain` and `kin` sampled by the encoder on rising edge `n`—is output shortly after rising edge `n+2`, and is available to be sampled on the rising edge of clock cycle `n+3`. (See Figure 3–5 on page 3–7). To enable cascaded encoding, the data paths fed by the `rdforce` and `rdin` inputs are not pipelined. Because `rdforce` and `rdin` are normally only used in cascaded configurations, this should not be a problem. In cases where the `rdforce` and `rdin` inputs are to be used in noncascaded configurations, they should be delayed two clock cycles with respect to their corresponding `datain` and `kin` values.

When the **register inputs/outputs** parameter is turned off, the encoder takes one clock cycle to encode a character. The encoded value—corresponding to the values of `datain` and `kin` sampled by the encoder on rising edge `n`—is output shortly after rising edge `n`, and is available to be sampled on the rising edge of clock cycle `n+1`. (See Figure 3–6).

Figure 3–5. Encoder Timing Diagram—Three Cycle Latency

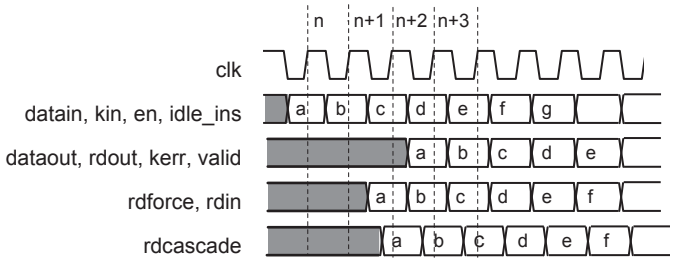
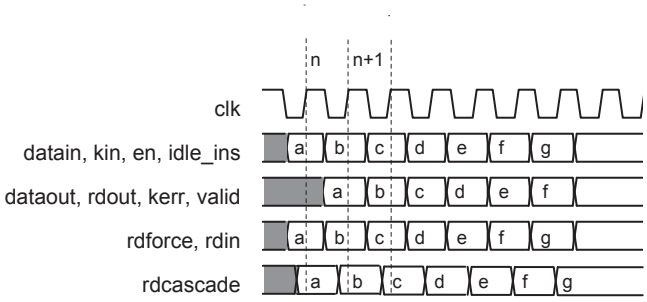


Figure 3–6. Encoder Timing Diagram—One Cycle Latency



Fibre Channel and IEEE 802.3z 1000BaseX

In Fibre Channel and IEEE 802.3z 1000BaseX applications the encoder does not automatically select the correct 8-bit data for Fibre Channel EOF or 1000BaseX Idle ordered sets. The running disparity based selection of the correct 8-bit data must be made before passing the data to the encoder.

Decoder

Data and identified 10-bit special *K* codes are converted from 10 bits to 8 bits; see Table 3–1 on page 3–4 for a list of the valid *K* codes, and Figure 3–1 on page 3–1 for an illustration of the conversion process.

When special 10-bit *K* codes are received, the special *K* codes are translated to 8-bit values, and the `kout` signal is asserted. The decoder also checks for invalid 10-bit codes.

When the decoder receives an invalid code, it asserts the `kerr` signal and decodes the value to an arbitrary number.

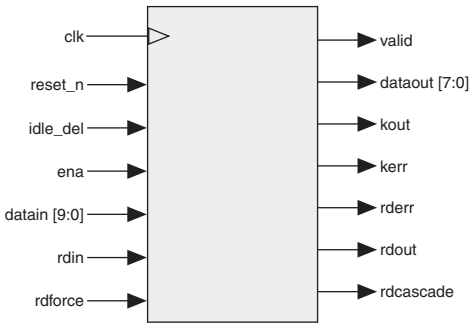
The decoder flags the 10B_ERR characters as invalid codes and asserts the `kerr` signal.

When the `idle_del` signal is asserted, it deletes all 10-bit words identified as the special IDLE character of K28.5.

When the receiver detects a disparity error, the `rderr` signal is asserted.

Figure 3–7 shows a block diagram of the decoder.

Figure 3–7. Decoder



Cascaded Decoding

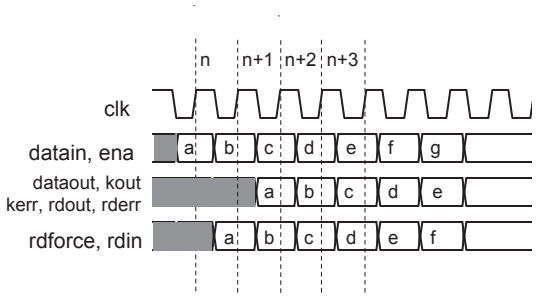
Two decoders can be cascaded to decode two words simultaneously. The decoders are cascaded—in a similar fashion as the encoders—by connecting the `rdcascade` output of the first decoder to the `rdin` input of the second decoder, and by connecting the `rdout` output of the second decoder to the `rdin` input of the first decoder. The `rdforce` inputs of both decoders must be tied high.

To enable cascaded decoding, the data paths fed by the `rdin` and `rdforce` inputs are not pipelined. If these inputs are to be used in non-cascaded decoders, they should be delayed by one clock cycle with respect to their corresponding `datain` and `kin` inputs.

Decoding Latency

The decoder is pipelined, thus it takes two clock cycles for a character to be decoded. The decoded value—corresponding to the value of `datain` sampled by the decoder on rising edge `n`—is output shortly after rising edge `n+1`, and is available to be sampled on the rising edge of clock cycle `n+2`. (See Figure 3–6 on page 3–7).

Figure 3–8. Decoder Timing Diagram



OpenCore Plus Time-Out Behavior

OpenCore® Plus hardware evaluation can support the following two modes of operation:

- *Untethered*—the design runs for a limited time
- *Tethered*—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely

All megafunctions in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior may be masked by the time-out behavior of the other megafunctions.

For MegaCore functions, the untethered timeout is 1 hour; the tethered timeout value is indefinite.

Your design stops working after the hardware evaluation time expires, and the following events occur:

- For the encoder:
  - The ena input signal is forced low (deasserted)
  - The dataout output is forced to the k28.5 pattern
  - The valid output is forced low (deasserted)
- For the decoder:
  - The ena input signal is forced low (deasserted)
  - The dataout output is forced to all zeros
  - The valid output is forced low (deasserted)

For more information on OpenCore Plus hardware evaluation, see "OpenCore Plus Evaluation" on page 1–2 and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Parameters

Table 3–2 shows the 8B10B Encoder/Decoder function parameters, which can only be set in the MegaWizard Interface (see "Parameterize" on page 2–6).

Table 3–2. 8B10B Encoder/Decoder Parameters

Parameter	Value
Mode of operation	Encoder or Decoder
Register inputs/outputs	On for a three cycle latency. Off for a one-cycle latency.

Signals

Tables 3–3 and 3–4 show the encoder and decoder signals.

Table 3–3. Encoder Signals (Part 1 of 2)

Signal Name	Direction	Description
clk	Input	Clock. The input is latched, and the result is output on this clock. There is a one clock cycle latency between the input and output.
reset_n	Input	Active low, reset. Asynchronously resets all registers in the MegaCore function. This signal should be deasserted synchronously to the rising edge of clk.
kin	Input	Command byte indicator. When high, indicates that the input is a command byte, not a data byte.
ena	Input	Enable encoder signal. When high, indicates that the data currently present on the datain input is to be encoded.

Table 3–3. Encoder Signals (Part 2 of 2)

Signal Name	Direction	Description
idle_ins	Input	Idle character insert. When high, idle (K28.5) characters are inserted when ena is not asserted.
datain[7:0]	Input	Data input. This is the 8-bit input word, data or command.
rdin	Input	Running disparity input. When rdforce is high, the value on this pin is used as the current running disparity instead of the internally generated one.
rdforce	Input	Force running disparity. When high, the rdin value overrides the internally generated running disparity.
kerr	Output	Special *K* character error. This signal is set high when ena and kin are high and the value on datain is not a valid special *K* character.
dataout[9:0]	Output	Data output. This is the 10-bit encoded output.
valid	Output	Valid signal. When high, indicates that a valid encoded word is present on the dataout output.
rdout	Output	Running disparity output. The current running disparity (after encoding the word present on the dataout output).
rdcascade	Output	Cascaded running disparity. Used when encoders are cascaded.

Table 3–4. Decoder Signals (Part 1 of 2)

Signal Name	Direction	Description
clk	Input	Clock. The input is latched, and the result output on this clock. There is a three clock cycle latency between the input and output.
reset_n	Input	Active low, reset. Asynchronously resets all registers in the MegaCore function. This signal must be deasserted synchronously to the rising edge of clk.
idle_del	Input	Idle delete signal. When high, idle words (K28.5) are removed from the stream (i.e. valid is set low when idle words are received).
ena	Input	Enable decoder signal. When high, indicates that the data currently present on the datain input is to be decoded.
datain[9:0]	Input	Data input. This is the 10-bit encoded input word.
rdin	Input	Running disparity input. When rdforce is high, the value on this pin is used as the current running disparity instead of the internally generated one.
rdforce	Input	Force running disparity. When high, the rdin value overrides the internally generated running disparity.

Table 3–4. Decoder Signals (Part 2 of 2)

Signal Name	Direction	Description
valid	Output	Valid signal. This signal is asserted when ena is asserted and new, non-idle data is present on dataout, even if it is the result of an illegal codeword. If an illegal codeword is received, kerr is also asserted. valid is also asserted for idle characters (K28.5) when ena is asserted and idle_del is not asserted.
dataout[7:0]	Output	Data output. This is the 8-bit decoded data or command.
kout	Output	Command output. When high, indicates that the output is a command byte, not a data byte.
kerr	Output	Special *K* error. Asserted high when an invalid 10-bit word is received, or when a 10B_ERR character is received.
rderr	Output	Running disparity error. When high indicates the running disparity rules have been violated.
rdout	Output	Running disparity output. The current running disparity (after decoding the word present on the dataout output).
rdcascade	Output	Cascaded running disparity. Used when decoders are cascaded.

This datasheet has been downloaded from:

www.EEworld.com.cn

Free Download

Daily Updated Database

100% Free Datasheet Search Site

100% Free IC Replacement Search Site

Convenient Electronic Dictionary

Fast Search System

www.EEworld.com.cn

All Datasheets Cannot Be Modified Without Permission

Copyright © Each Manufacturing Company